07-07-00

Rec'd PCT/PTO 06 JUL 2000

OIPE JUL 07 2000 PATENT & TRADEMARK OFFICE

FORM PTO-1390 (Modified) REV 10-95)

U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371

ATTORNEY'S DOCKET NUMBER	P-5857
U.S. APPLICATION NO. (IF KNOWN, SEE 37 CFR	09/582939

INTERNATIONAL APPLICATION NO.	INTERNATIONAL FILING DATE	PRIORITY DATE CLAIMED
PCT/FR99/02696	November 4, 1999	November 6, 1998

TITLE OF INVENTION DATA COMPACTION METHOD FOR AN INTERMEDIATE OBJECT CODE PROGRAMME EXECUTABLE IN AN ONBOARD SYSTEM PROVIDED WITH DATA PROCESSING RESOURCES AND CORRESPONDING ONBOARD SYSTEM WITH MULTIPLE APPLICATIONS

APPLICANT(S) FOR DO/EO/US

Ulrik Pagh SCHULTZ; Gilles MULLER; Charles CONSEL; Lars CLAUSEN and Christian GOIRE

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☒ This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.
2. ☐ This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. ☐ This is an express request to begin national examination procedures (35 U.S.C. 371(f)) at any time rather than delay examination until the expiration of the applicable time limit set in 35 U.S.C. 371(b) and PCT Articles 22 and 39(1).
4. ☐ A proper Demand for International Preliminary Examination was made by the 19th month from the earliest claimed priority date.
5. ☒ A copy of the International Application as filed (35 U.S.C. 371 (c) (2))
   a. ☒ is transmitted herewith (required only if not transmitted by the International Bureau).
   b. ☐ has been transmitted by the International Bureau.
   c. ☐ is not required, as the application was filed in the United States Receiving Office (RO/US).
6. ☒ A translation of the International Application into English (35 U.S.C. 371(c)(2)).
7. ☒ A copy of the International Search Report (PCT/ISA/210).
8. ☐ Amendments to the claims of the International Application under PCT Article 19 (35 U.S.C. 371 (c)(3))
   a. ☐ are transmitted herewith (required only if not transmitted by the International Bureau).
   b. ☐ have been transmitted by the International Bureau.
   c. ☐ have not been made; however, the time limit for making such amendments has NOT expired.
   d. ☐ have not been made and will not be made.
9. ☐ A translation of the amendments to the claims under PCT Article 19 (35 U.S.C. 371(c)(3)).
10. ☐ An oath or declaration of the inventor(s) (35 U.S.C. 371 (c)(4)).
11. ☐ A copy of the International Preliminary Examination Report (PCT/IPEA/409).
12. ☐ A translation of the annexes to the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371 (c)(5)).

Items 13 to 18 below concern document(s) or information included:

13. ☐ An Information Disclosure Statement under 37 CFR 1.97 and 1.98.
14. ☐ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.
15. ☒ A **FIRST** preliminary amendment.
    A **SECOND** or **SUBSEQUENT** preliminary amendment.
16. ☐ A substitute specification.
17. ☐ A change of power of attorney and/or address letter.
18. ☒ Certificate of Mailing by Express Mail
19. ☒ Other items or information:

PCT Request (5 pgs.)
9 Sheets of Formal Drawings
Verification of English Translation (1 pg.)
Return Receipt Postcard

10 Rec'd PCT/PTO 06 JUL 2000

U.S. APPLICATION NO (IF KNOWN, SEE 37 CFR	INTERNATIONAL APPLICATION NO.	ATTORNEY'S DOCKET NUMBER
09/582939	PCT/FR99/02696	P-5857

20. The following fees are submitted:.

BASIC NATIONAL FEE (37 CFR 1.492 (a) (1) - (5)) :

		CALCULATIONS	PTO USE ONLY
☒ Search Report has been prepared by the EPO or JPO	$840.00		
☐ International preliminary examination fee paid to USPTO (37 CFR 1.482)	**$720.00**		
☐ No international preliminary examination fee paid to USPTO (37 CFR 1.482) but international search fee paid to USPTO (37 CFR 1.445(a)(2))	**$790.00**		
☐ Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2) paid to USPTO	**$1,070.00**		
☐ International preliminary examination fee paid to USPTO (37 CFR 1.482) and all claims satisfied provisions of PCT Article 33(2)-(4)	**$98.00**		
ENTER APPROPRIATE BASIC FEE AMOUNT =		**$840.00**	
Surcharge of **$130.00** for furnishing the oath or declaration later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492 (e)).		**$0.00**	

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE		
Total claims	11 - 20 =	0	x $18.00	**$0.00**	
Independent claims	3 - 3 =	0	x $78.00	**$0.00**	
Multiple Dependent Claims **(check if applicable)**.			☐	**$0.00**	
TOTAL OF ABOVE CALCULATIONS =				**$840.00**	
Reduction of 1/2 for filing by small entity, if applicable. Verified Small Entity Statement must also be filed (Note 37 CFR 1.9, 1.27, 1.28) **(check if applicable)**.			☐	**$0.00**	
SUBTOTAL =				**$840.00**	
Processing fee of **$130.00** for furnishing the English translation later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492 (f)). +				**$0.00**	
TOTAL NATIONAL FEE =				**$840.00**	
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31) **(check if applicable).**			☐	**$0.00**	
TOTAL FEES ENCLOSED =				**$840.00**	
				Amount to be: refunded	$
				charged	$

☐ A check in the amount of to cover the above fees is enclosed.

☒ Please charge my Deposit Account No. **18-2284** in the amount of **$840.00** to cover the above fees. A duplicate copy of this sheet is enclosed.

☒ The Commissioner is hereby authorized to charge any fees which may be required, or credit any overpayment to Deposit Account No. **18-2284** A duplicate copy of this sheet is enclosed.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137(a) or (b)) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:

Michael L. Kenaga
PIPER MARBURY RUDNICK & WOLFE
P.O. Box 64807
Chicago, Illinois, 60664-0807

(312) 368-4000

Michael L. Kenaga
SIGNATURE

Michael L. Kenaga
NAME

34,639
REGISTRATION NUMBER

July 6, 2000
DATE

09/582939

10 Rec'd PCT/PTO 06 JUL 2000

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:	SCHULTZ et al.	)	**PATENT**
Serial No.:	National Phase of International Patent Application PCT No. FR99/02696	)	Atty. Docket No. P-5857
		)	Examiner: Unassigned
Filed:	HEREWITH	)	Group Art Unit: Unassigned
For:	DATA COMPACTION METHOD FOR AN INTERMEDIATE OBJECT CODE PROGRAMME EXECUTABLE IN AN ONBOARD SYSTEM PROVIDED WITH RESOURCES DATA PROCESSING AND CORRESPONDING ONBOARD SYSTEM WITH MULTIPLE APPLICATIONS	)	

PRELIMINARY AMENDMENT

Commissioner of Patents and Trademarks
Washington, D.C. 20231

Dear Sir:

Prior to examination of this application, please amend the above-identified application as follows:

IN THE CLAIMS:

Please cancel claims 1-11 without prejudice and substitute therefore the following new claims:

REMARKS

Entry of the above amendment is respectfully requested.

The new claims 1-11 are substantially similar to original claims 1-11 set out in the PCT International Application, with the exception that the claims have been amended to conform to U.S. practice. The Examiner is kindly requested to renumber the new claims 1-11 as claims 12-22 respectively, and to change the dependency accordingly.

Respectfully submitted,
PIPER MARBURY RUDNICK & WOLFE

Date: July 6, 2000

By: Michael L. Kenaga
Michael L. Kenaga
Reg. No. 34,639

PIPER MARBURY RUDNICK & WOLFE
P.O. Box 64807
Chicago, IL 60664-0807
(312) 368-4000

CLAIMS

1. A method of compacting an intermediate programme consisting of a sequence of standard instructions, used in an on-board system, said on-board system being provided with a memory and a programme language interpreter capable of turning the intermediate programme into instructions of an object code that can be run directly by a microprocessor, said method consisting in:

a) searching through said intermediate programme for identical sequences of successive standard instructions;

b) subjecting said identical sequences of successive instructions to a comparison test to find a function, based on at least the number of occurrences of these sequences in said intermediate programme, that is higher than a reference value and, if the test returns a positive response, for each identical sequence of successive standard instructions which satisfies said test step,

c) generating a specific instruction by defining a specific operating code and associating said specific operating code with the sequence of successive standard instructions which satisfied said test,

d) replacing each occurrence of each sequence of standard successive instructions in said intermediate programme with said specific operating code associated with it to obtain a compacted intermediate programme, consisting of a series of standard instructions and specific operating codes, and

e) storing in said memory an execution table which enables a reciprocal link to be established between each specific operating code inserted and the sequence of successive standard instructions associated with the latter, thereby enabling the memory space occupied by said compacted intermediate programme to be optimised by storing only one occurrence of said identical sequences of successive standard instructions in said memory.

2. A method as claimed in claim 1, wherein said function is also a function of the size of each identical sequence of successive instructions.

3. A method as claimed in claim 1, wherein in order to compress a plurality of intermediate programmes, said method also consists in:

- storing said execution table relating to at least one compacted intermediate programme and, for every additional intermediate programme subjected to a compaction process,

- reading said stored execution table and

- running the compaction process for every additional programme, taking account of the specific codes and instructions stored in said execution table.

4. A method of running a compacted intermediate programme obtained by applying a compaction method as claimed in claim 1, said compacted intermediate programme consisting of a succession of standard instructions and specific operating codes stored in the memory of an on-board system, wherein said method consists in:

- recognising in said memory the existence of a stored execution table containing at least one sequence of successive instructions associated with a specific operating code by means of a reciprocal link;

- calling up a command, via said programme language interpreter, to read the successive standard instructions or specific operating codes of said compacted intermediate programme and, in the presence of a specific operating code:

- retrieving said sequence of successive instructions associated with said specific operating code from the memory by means of a read instruction and, in the presence of a standard instruction,
- commanding the execution of said standard instruction by means of a read instruction,

5. A method as claimed in claim 4, wherein if a sequence of successive instructions associated with a specific operating code is called up, the current value of a programme counter is incremented in a stack associated with the specific operating codes and a programme pointer points to the first instruction of said sequence of specific instructions, after which, on running an instruction to end the sequence of specific instructions, said programme counter is decremented and the execution process continues starting with the next instruction or specific operating code.

6. A method as claimed in claim 5, wherein the stack associated with the specific operating codes and the stack associated with the standard instructions are a single stack.

7. A multi-application on-board system comprising computing resources, a memory and language interpreter capable of turning an intermediate programme into instructions which are directly executable by the computing resources, wherein said multi-application on-board system also at least comprises:

- one table of standard codes constituting said intermediate programme stored at the level of said language interpreter;

- at least one compacted intermediate programme constituting an application and consisting of a series of specific instruction codes and standard instruction codes, said specific instruction codes corresponding to sequences of successive standard instructions;

- an execution table enabling a reciprocal link to be established between an operating specific instruction code and the sequence of successive standard instructions associated with the latter, said at least one compacted intermediate programme and said execution table being stored in said memory, thereby enabling the memory space occupied by said compacted intermediate programme to be optimised by storing in said programmable memory only one occurrence of said identical sequences of successive instructions.

8. An on-board system as claimed in claim 7, wherein said execution table comprises at least:

- a file of successive sequences corresponding to said specific instruction codes;

- a table of specific instruction codes and addresses at which said specific instruction codes are embedded in the table of successive sequences.

9. An on-board system as claimed in claim 8, wherein said file of successive sequences corresponding to said specific instruction codes and said table of specific instruction codes are stored in a programmable memory of said on-board system.

10. A compaction system for an intermediate programme, said intermediate programme consisting of a series of standard instructions which can be executed by a target unit, wherein said system comprises at least:

- means for analysing all the standard executable instructions enabling, by means of a reading process, said intermediate programme to distinguish between and establish a list of all the sequences of executable standard instructions contained in said intermediate programme;

- means for counting the number of occurrences in this intermediate programme of each of the sequences of executable standard instructions forming part of said list;

- means for allocating to at least one sequence of executable standard instructions a specific code associated with this sequence of executable standard instructions in order to generate a specific instruction;

- means for replacing, in said intermediate programme, each occurrence of said sequence of executable standard instructions with said specific code associated with this sequence of executable standard instructions, representative of said specific instruction, thereby enabling a compacted programme to be generated comprising a succession of executable standard instructions and specific instructions.

11. A compaction system as claimed in claim 10, wherein said means for allocating to at least one sequence of executable standard instructions a specific code associated with said sequence of executable standard instruction in order to generate a specific instructions comprises at least:

- means for computing the value of a function based on at least the length of and number of occurrences of said sequence of executable standard instructions, said function being representative of the compression gain for said sequence of executable standard instructions;

- means for comparing the value of said function with a threshold value and, if said comparison returns a positive response,

- means for writing to a file, with a reciprocal link, a specific code and this sequence of executable standard instructions in order to constitute said specific instruction.

09/582939

10 Rec'd PCT/PTO 06 JUL 2000

A DATA COMPACTION METHOD FOR AN INTERMEDIATE OBJECT CODE PROGRAMME EXECUTABLE IN AN ON-BOARD SYSTEM PROVIDED WITH DATA PROCESSING RESOURCES AND CORRESPONDING ONBOARD SYSTEM WITH MULTIPLE APPLICATIONS

The present invention relates to a method of compacting a programme of the intermediate object code type, which can be run in an on-board system provided with data processing resources, and to the corresponding compaction method.

These days, on-board systems provided with data processing resources enable increasingly complex and larger numbers of functions to be performed because the hardware used for these portable objects and their software, or more specifically the application programmes embedded in the latter as a means of enabling them to run one or more specific functions, are constantly being optimised. The concept of on-board system covers any portable computer system, such as a portable object, microprocessor card or similar, as opposed to a conventional microcomputer.

This is particularly so in the case of microprocessor cards, also known as chip cards, such as that illustrated in figure 1a, used in conjunction with a compiler to generate instructions and an interpreter which enables these instructions to be run by the microprocessor, as illustrated in figure 1b. Conventionally, as illustrated in figure 1a, a microprocessor card 10 comprises an input/output system 12 linked to the microprocessor 14, a RAM memory 16, a non-volatile memory 18 comprising a read only ROM memory 18b and a programmable memory 18a. All of these elements are linked to the microprocessor 14 by a BUS line. A data encryption/decryption module 20 may be provided, as required.

Figure 1c illustrates all the application software elements, such as electronic purse, electronic commerce or health, embedded in the non-volatile programmable memory, the interpreter in a non-volatile programmable memory or a read only memory and the operating system in a read only memory ROM.

The intermediate object code is generated by the compiler from a source programme, usually written in high level language based on ASCII characters. The source programme and the corresponding intermediate object code may be run by all the standard microprocessors because the interpreter ensures that the standard instructions of the intermediate object code are translated, in software terms, into instructions that can be directly executed by the microprocessor.

By way of example, although this is not restrictive, microprocessor card manufacturers have recently developed interpreters embedded in the read only memory ROM. This type of interpreter reads in sequence a programme or intermediate object code, supporting an application for example, which is loaded into the programmable memory of the microprocessor card, for example. Each standard instruction of this intermediate object code is interpreted by the interpreter, then run by the microprocessor. As a general rule, the standard instructions of the intermediate object code enable changing functions to be processed, such as arithmetic processing and object manipulation. The object concept relates to computed objects such as lists, data tables or similar.

However, due in particular to the fact that these microprocessor cards are portable in nature, the size and space occupied by the latter are limited. The same applies to the size of their programmable memory, which is limited, by construction, to several kilobytes. This structural limitation makes it impossible to run large application programmes.

Furthermore, the current tendency to use multi-application on-board systems is faced with the recurring problem caused by the fact that the number of applications installed on a same on-board system or microprocessor card rarely exceeds three applications.

The objective of this invention is to overcome the disadvantage outlined above by running a method for compacting a programme of the intermediate object code type, which can be used in an on-board system of the microprocessor type in order to free up memory space in the programmable memory of this on-board system, thereby enabling at least one additional application to be embedded once the latter has been compacted.

Another objective of this invention is to implement a system of compacting programmes of the intermediate object code type which will allow a programme of the compacted intermediate object code type to be embedded in a multi-application, on-board system having data processing resources that will allow compacted programmes of the intermediate object type to be run without significantly altering the running time and to do so in total transparency with regard to the process inherent in each non-compacted application.

The method of compacting a programme of the intermediate object type proposed by the invention, which consists of a sequence of standard instructions, this on-board system being provided with a memory and a programme language interpreter capable of turning intermediate object code into instructions of an object code that can be run directly by a microprocessor and this programme normally being stored in the memory of this on-board system, is remarkable because the intermediate object code programme is searched in order to find identical sequences of successive standard instructions and the identical sequences of successive standard instructions are subjected to a comparison test to find a function, based on at least the number of occurrences of these sequences in the intermediate object code programme, that is higher than a reference value. If the above-mentioned test returns a positive response, a specific instruction is generated for each identical sequence of successive standard instructions which satisfies the test step, by defining a specific operating code and associating this specific operating code with the sequence of successive standard instructions. In addition, each occurrence of each sequence of standard successive instructions in the stored intermediate object code programme is replaced by the specific operating code associated with it to obtain a compacted intermediate object code programme, a series of standard instructions and specific operating codes. A decompression table is stored in the memory which enables a reciprocal link to be made between each specific operating code inserted and the sequence of successive standard instructions associated with the latter. This process enables the memory space occupied by the compacted intermediate object programme to be optimised by storing only one occurrence of identical sequences of successive standard instructions in the programmable memory.

The method, the system of compacting an intermediate object code programme and the corresponding multi-application on-board system proposed by this invention may be used in the technical field of on-board systems, more specifically for running and managing microprocessor cards.

They will be more readily understood from the description below and the appended drawings, in which, apart from figures 1a to 1c which illustrate the prior art,

- figure 2a is a general flow chart illustrating a method of compacting an intermediate object code programme as proposed by this invention;

- figure 2b is a synoptic diagram illustrating how the different operators needed to obtain a compacted intermediate object code programme and parameters enabling this programme to be decompressed or executed, are applied;

- figure 2c, given purely as an illustration, shows this compacted intermediate object code programme embedded in a programmable non-volatile memory of a microprocessor card and the parameters used to decompress and run the latter;

- figure 3a is a diagram of one specific embodiment, although this is not restrictive, of the structure of a first file made up of the parameters for running or decompressing this intermediate object code programme;

- figure 3b is a diagram showing one specific embodiment, although this is not restrictive, illustrating the structure of a second file made up of these parameters for running or decompressing this intermediate object code programme;

- figure 4 shows, as an illustrative example, a compacted intermediate object code programme, as proposed by the invention, embedded in a non-volatile programmable memory of a microprocessor card or on-board multi-application system;

- figure 5 shows an illustrative example of a specific method of compacting an intermediate object code programme in which specific codes, relating to separate applications or intermediate object code programmes, are updated;

- figures 6a and 6b illustrate, in the form of functional elements, a system for compacting an intermediate object code programme as proposed by this invention.

The method of compacting an intermediate object code programme as proposed by this invention will now be described with reference to figure 2a. The terms intermediate object code programme cover any intermediate programme for the purpose of this patent application.

This method will be described with reference to running an on-board system consisting of a microprocessor card of the type illustrated in figure 1a for example, although this example is not restrictive, this intermediate object code programme being obtained in a conventional manner, as illustrated in figure 1b, a plurality of applications embedded in the programmable memory of the interpreter and the operating system OS in the ROM memory being illustrated in figure 1c, although again this is not restrictive.

The intermediate object code programme consists of a series of standard instructions which can be run by the microprocessor through the interpreter.

The method of compacting a programme of this type consists in firstly embedding the latter in the programmable memory 18a, searching the intermediate object code programme at a step 1000 as illustrated in figure 2a for identical sequences of successive standard instructions, these identical sequences being shown by Si. By identical sequences is meant a sequence of a given number n of octets likely to appear on a repetitive basis in the above-mentioned intermediate object code programme. Accordingly, the rank i of identical sequences indicates separate sequences for different values of i. Furthermore, the search step 1000 mentioned above consists in determining the number of occurrences N_1 of each said identical sequence S_i. At the end of the search step 1000, a plurality of identical sequences S_i will have been found, each sequence S_i being separate, and a number N_i representing the number of occurrences of each of the sequences S_i in the intermediate object code programme.

After said step 1000, the compacting method proposed by the invention consists in subjecting, at a step 1001, the identical sequences of successive standard instructions S_i to a comparison test of a function $f(N_i)$ based on at least the number of occurrences N_i associated with an identical sequence S_i. In figure 2a, the comparison test is written:

$$f(N_i) > \text{Ref}.$$

If the response to the test 1001 is negative, in which case the function of at least the number of occurrences N_i is not greater than the reference value, the test 1001 is applied to the next identical sequence, of rank i+1, the index i being incremented at step 1002.

The steps 1000, 1001 and 1002 illustrated in figure 2a therefore enable a search to be run in the intermediate object code programme for identical sequences or series of octets or, at the very least, a given significant number of these identical sequences, as will be described farther on in the description.

If the response to said test 1001 is positive, the compacting method proposed by the invention then consists in generating a specific instruction, denoted by IS_i, by defining a specific operating code, denoted by C_i, and associating with this specific operating code the sequence of successive standard instructions which satisfied the test, this being the sequence of successive standard instructions S_i. In figure 2a, the step by which specific instructions are set up is written:

$$IS_i = C_i:S_i.$$

It should be pointed out that the step of defining a specific operating code and associating it with the sequence of successive standard instructions S_i may consist in assigning a code value and associating this code value with said sequence of instructions S_i in the form of a list, for example.

After step 1003, the compacting method then moves on to step 1004, at which each occurrence of the sequence of successive standard instructions S_i in the intermediate object code programme is replaced by the specific operating code C_1 which is associated with it in order to obtain a compacted object code programme, denoted by FCC, which is a succession of standard instructions and specific operating codes C_1.

The replacement process can then be reiterated for each identical sequence or series of standard instructions S_i as long as the index i is lower than a number P of identical sequences, whereby a comparison test 1005 of the index i with the value P will prompt a return to step 1002 at which the index i described above is incremented whenever the response to this test is positive.

In particular, it should be pointed out that by iterating the replacement process set up in this manner, a compacted object code programme will be obtained with which an execution file for the latter is associated, this file being denoted by FEX, the execution file consisting of at least one reciprocal match between each specific code C_i and said sequence of successive standard instructions S_i.

Once the two above-mentioned files have been obtained, being the compacted intermediate object code programme and execution file, on a negative response to test 1005 for example, a storage process can be initiated, whereby said intermediate object code programme FCC obtained and of course the execution file FEX described above are stored in the programmable memory 18a, for example. Said storage may be in the non-volatile memory 18, programmable memory 18a or even in the read only memory 18b, although this is not restrictive.

As far as said comparison test 1001 is concerned, it should be pointed out that the function of at least the number of occurrences of each identical sequence S_i may be defined so as to optimise the gain in compression thus achieved. In one embodiment, which is not restrictive, this function may be set up so that a comparison is made between the size of each identical sequence of successive standard instructions and a threshold value, expressed as a number of standard instructions, for example.

Figure 2b provides an illustrative example of an operating mode which allows a compacted intermediate object code programme to be generated using the method proposed by the invention.

During an initial stage, the creator of the intermediate object code programme sets up a text file containing the source programme. This programme, put together by the latter on the basis of an evolved language, is generally written in ASCII code so that it can be easily read and can contain comments to facilitate understanding on the one hand and development of the latter on the other. The source programme thus created is inserted in a compiler of the conventional type, known as a standard compiler, the purpose of which is to transform each programme line into executable instructions or, at the very least, into instructions which can be interpreted in order to obtain an intermediate object code programme consisting of a standard sequence of instructions that can be interpreted by the interpreter.

The intermediate object code file thus obtained after the compilation process is placed in a compression system allowing the compacting method to be run as described above with reference to figure 1. This compression system will be explained later in the description.

The compaction process applied and described above will then produce a file of interpretable instructions FCC, i.e. the file constituting the compacted intermediate object code programme and the execution file FEX mentioned earlier in the description.

The operating mode of the compression system will be described below with reference to a specific example.

Firstly, the compactor system analyses all the standard instructions I_s and draws up a list of all the series of standard instructions existing in the file forming this latter.

If said file contains 1000 octets, for example, the compactor system launches a search procedure on all the series of at least two octets up to a number Q, for example. Said search may be run for series of two octets, then three octets and so on up to Q octets. In a preferred embodiment, the value of the number Q was 500.

Consequently, for every sequence of instructions S_i, made up of a series of standard instructions I_s, the compactor system determines whether this sequence S_i is already contained in the list. If such is the case, the compactor system adds one unit to the number of occurrences N_i of said sequence S_i.

By the end of said search process, the compactor system will have generated a complex list containing all the instruction sequences S_i examined, each sequence having a number of occurrences N_i in the relevant intermediate object code programme assigned to it.

An illustrative table is given below for an intermediate object code programme made up of the following series of instructions:

1-7-3-5-7-3-7-3-5-7.

Whereas for the purpose of the illustration given in the table, TABLE I below, said series of instructions contains ten instructions, each instruction being represented by one octet and illustrated by a digit from 1 to 7, the successive sequences of instructions examined comprise 2, 3, 4 then 5 octets.

The successive instruction sequences S_i which occur in said intermediate object code programme a number of times greater than or equal to two, are given in the table below.

TABLE 1

4 octets	[7-3-5-7]:2		
3 octets	[7-3-5]:2	[3-5-7]:2	
2 octets	[7-3]:3	[3-5]:2	[5-7]:2

Secondly, the compactor system replaces certain sequences S_i of TABLE I by a code denoting specific instructions.

The code for specific instructions C_i is determined chronologically on the basis of the first code corresponding to a standard instruction. In a commonplace intermediate object code, there are currently 106 standard instructions and the codes for these instructions fall between 000 and 105. The first specific instruction code C_i can then have the value 106, the second value 107 and so on. Whenever the identical sequences of instructions S_i are replaced by a new specific instruction code C_i, once such an operation has been completed, the list illustrated in the table above is then re-computed.

By way of non-restrictive example and in the case where the 4-octet sequence of instructions, the sequence 7-3-5-7, shown in the table above is replaced and a corresponding code 106 is allocated, the compacted intermediate object code programme becomes:

1=106-3-106.

Under these conditions, a sequence of standard instructions I_s and specific instructions IS existing in identical form at least twice no longer exist. Clearly, the file constituting the compressed intermediate object code programme FCC and the execution or decompression file for this latter are stored at the level of the compactor system mentioned above.

Once the compacting operation has been performed by the compactor system, there exists an intermediate object code programme, strictly speaking, which can be executed by the target system, and said execution file FEX. The former contains the standard instructions I_s and the specific instructions IS whilst the latter comprises at least one table allowing the specific codes C_i to be linked with the sequences of standard instructions S_i replaced by said specific codes. Clearly, these two files may be regrouped into one and the same file with a view to transferring the latter to the intended target system. i.e. the microprocessor card designed to receive it.

With regard to the execution file FEX, it should be pointed out that it contains at least one file, shown by MEM-SEQ, made up of a succession of several fields such as a specific code C_i field, a sequence field S_i, as outlined above.

Following said operation, the single file or, as applicable, the two above-mentioned files, are transmitted to the target system and processed directly by a loading programme. This loading programme is mainly responsible for writing the data received to the programmable memory 18a or the read only memory 18b so that it can effectively be run subsequently.

As a non-restrictive example, the file relating to the compacted intermediate object code programme FCC is stored, without processing, in said programmable memory 18a starting at a given address, denoted by ADR-MEM-PGM.

As for the execution file FEX, the loading programme analyses the data of this file and dynamically creates a table, denoted by TAB-PRO, enabling the specific instruction codes C_i to be associated with the series of instructions. In fact, the table TAB-PRO enables a reciprocal match to be made between said specific instruction codes C_i and an embedding address which enables the corresponding instructions to be run.

Figure 2c illustrates the embedding on the one hand of the support file for the compacted intermediate object code programme FCC, the execution file FEX and the file TAB-PRO mentioned above, this latter file having been created by the loading programme in the programmable memory 18a of the microprocessor card.

In this drawing, whilst the table of codes for standard instructions I_s is stored at the level of the interpreter in a table TAB-STD, it is in the programmable memory 18a that the execution file FEX and the file TAB-PRO are stored, enabling the address skips to be linked to the corresponding specific instruction codes C_i, these two tables therefore enabling the compacted intermediate object code programme FCC to be run effectively at the level of the microprocessor of the target unit. An executable unit is therefore provided which can be run by the interpreter under conditions that will be explained blow.

Before moving on to explain how a compacted intermediate object code programme FCC is run, a detailed description will be given of the structure of the execution files FEX and the file TAB-PRO and the functional relationship between these latter, with reference to figures 3a and 3b.

Figure 3a provides a detailed illustration of the execution file FEX, which, as explained above, has, in addition to the fields for specific codes C_i and instruction sequences S_i, a field to denote the end of macro-instructions, denoted by FM, indicating the end of said sequence. In a preferred although not restrictive embodiment, each specific code C_i may be inserted at the beginning of the field, on one octet for example, after which each corresponding sequence S_i is inserted in a second field of variable length. The end of macro code FM is of the standard type and corresponds to that used in the conventional languages mentioned earlier in the description.

When an execution file FEX is received whose data structure corresponds to that illustrated in figure 3a, for example, the different fields C_i, S_i and FM are processed separately.

Firstly, the specific code C_i of the corresponding specific instruction IS is written to the file TAB-PRO and the sequence of instructions S_i associated with this specific code constituting said specific instruction is written to a file or memory denoted by MEM-SEQ from an address denoted by ADR-1. This code C_i for the corresponding specific instruction is written to the address TAB-PRO + 3 x (CODE-106). In this relation, the address TAB-PRO is specified as being the address at which the file TAB-PRO is opened whilst the value CODE represents the numerical value of the corresponding code C_i. The operating mode illustrated in figure 3b corresponds to an address value TAB-PRO which is arbitrarily set at 0, the first specific code allocated having the value 106 and the following successive specific codes allocated having the values 107 and so on. Figure 3b shows only four specific codes 106, 107, 110 and 120 in order to facilitate understanding, the other memory spaces being filled with arbitrary values.

Under these conditions, Adr-i is the first available address in the memory MEM-SEQ, this address corresponding to the address Adr-1 for the first sequence of instructions $S_i = S_1$. From this first address, which constitutes the opening of the file in the memory MEM-SEQ, the sequences of instructions S_i are therefore written in sequence in the order in which they are loaded. The end of macro code FM is also written at the end of the corresponding series.

After said process of writing to the memory MEM-SEQ and after a step to check that the writing process has proceeded correctly, the loading programme writes to the table TAB-PRO after each specific code C_i the value of the address at which the sequence was written in the memory MEM-SEQ. The loading programme then re-computes a new write address for the next sequence S_i of rank i, incremented or decremented depending on the mode used to run through said sequences of instructions S_i.

An execution process of a compacted intermediate object code programme supported by a file FCC as described above and containing specific instructions will now be described with reference to figure 4.

Such a programme is run by means of the interpreter with the aid of an instruction pointer, denoted by PI. In fact, the instruction pointer PI reads the code of the instruction to be carried out, standard instruction Is or specific instruction IS, and applies this code to the interpreter which then triggers the actions corresponding to it.

At the start of running a programme, the instruction pointer PI is loaded with the address from which this programme starts, i.e. the address ADR-MEM-PGM.

The interpreter analyses the value of the code read by the instruction pointer PI. As part of this analysis, the latter determines whether this code value corresponds to a standard type code Cs or, if not, to a specific code C_i. This operation is run from the table TAB-STB stored at the level of the interpreter and by associating the standard instruction codes and hence the standard instructions Is with the execution addresses in its programme.

If the value of the code read is not in the latter table, the interpreter issues a call to read the table TAB-PRO in order to check for the existence of the code value read in the latter table. If the code read is no longer in this latter table, the interpreter will be unable to run the instruction read and the programme run will be halted, prompting an error message, not illustrated in the flow chart of figure 4.

In said figure 4, 2000 denotes the start of the execution operation, 2001 the operation by which the instruction pointer PI is initialised on the first programme instruction and 2002 an operation whereby the instruction indicated by the instruction pointer PI is read. This operation in fact corresponds to the process of reading said code value.

In the same manner, at step 2003 in figure 4, a check is run to determine whether the code value read in the table belongs or does not belong to the standard codes TAB-STB and whether this code value read belongs to the table TAB-PRO, this process in effect forming said test 2003, after which a distinction is made as to whether the instruction INS read is a standard instruction Is or a specific instruction IS. Fig. 4 does not illustrate the situation in which the code read does not belong as explained above and the other of the two tables which generates an error message ,so as not to overload the drawing.

If a positive response is received to said test 2003 and the code read corresponds to a specific instruction, the instruction pointer PI is computed and stored in the stack so that it can then move on to the next instruction. The interpreter reads from the table TAB-PRO the value of the address for the sequence of instructions S_i associated with the specific code C_i read and initialises the value of the instruction pointer PI with this value. All of these operations are shown under reference 2004 in figure 4. Following said step 2004, the interpreter loops back to the step of reading the code, as illustrated in figure 4, by returning to step 2002.

If a negative response is received to test 2003 and the code read corresponds to an instruction of the standard type Is, the interpreter will check in a test step 2005 whether the value for this code corresponds to an end of macro value representing an end of sequence. If such is the case, the value previously stored in the stack memory is extracted and the stack is updated, this value being loaded into the instruction pointer PI. The operation of extracting the value previously stored in the stack constituting a return address, followed by an update of the stack, is illustrated at 2006, the return address being denoted by ADR-RET. After said step 2006, the interpreter loops back to the process at the step where the code value is read, i.e. step 2002. If a negative response is received to test 2005 and the value of the code read corresponds to an instruction of the standard type but does not correspond to an end of macro or end of series, then the code is run by the interpreter in a manner known per se. However, as illustrated in figure 4, a test step 2007 is provided in this case prior to actually running said standard instruction. Test 2007 consists in checking that the value for the code and the corresponding instruction INS does not correspond to that for an end of programme. If a positive response is received to said test 2007, the execution step 2008 is then run for this instruction by the interpreter, this execution step being associated with a step by which the instruction pointer is incremented so that it will point to the next instruction. After said step 2008, the interpreter loops back to the step at which the code value indicated by the instruction pointer PI is read, i.e. the reading step 2002.

If a negative response is received to test 2007, in which case the instruction corresponds to an end of programme instruction, and end step 2009 is run. In this case, the interpreter halts its action and hands over to the operating system OS. It then waits for a new command instruction.

The embodiment and method of implementing the process of running a compacted intermediate object code programme as described with reference to figure 4 is not restrictive.

Firstly, it should be pointed out that the stack memory may be subdivided into two separate stack memories, one stack memory for the standard instructions Is and one stack memory for the specific instructions IS, alternatively referred to as macro instructions. In an embodiment of this type, the maximum number of specific instructions IS interleaved within the procedures is known. In order to obtain the total size occupied by this stack, it is sufficient to multiply by the maximum number of interleaved procedures. Using a separate stack memory for the specific instructions IS reduces the total memory consumption as compared with using a single stack.

Furthermore, in order to increase the number of specific instructions IS which may be used instead and in place of the limited number of specific instructions between 106 and 255, in the example described earlier in the description by way of illustration, the specific codes C_i can advantageously be encoded on two octets. Under these circumstances, a specific code value such as the value 255 will then indicate coding on two octets.

Finally, the target system, if it is an on-board multi-application system, comprises several compiled and compacted programmes, i.e. several files FCC of the type described above. These programmes must operate independently. This being the case and there being only one interpreter, it runs all the application programmes loaded by the loading programme. If two application programmes use specific instructions, in the embodiment explained earlier in the description, it is possible that the compactor system will assign the same specific code C_1 for two series of different instructions.

In order to remedy such a situation and to enable the interpreter to distinguish between the two codes, the fields for the execution file FEX as described above in relation to figure 3a may be supplemented by a third parameter relating to an identification number for the application in question. This identification number will then also be stored for each assigned specific code listed in the table TAB-PRO. This latter parameter is in fact the reference for the programme loaded at the same time as the file containing the table, allowing each specific instruction code C_i to be associated with instruction sequences S_i replaced by these latter for the application in question. When the interpreter is running the programme application, it will then be able to single out the specific instructions relating to this application.

Clearly, the process described above enabling a multi-application on-board system to be run has the disadvantage of increased memory consumption because an additional field has to be allocated to relate back to the application number in question.

A more advantageous process will now be described with reference to figure 5.

Figure 5 illustrates an on-board system such as a microprocessor card comprising several applications, shown by A_1 to A_k, the values A_1 to A_k in fact constituting identification numbers for each application. To this end, when compacting any source programme or application with a given identification number A_1 to A_k for example, in accordance with the method proposed by the invention as explained earlier in the description, the target system, i.e. microprocessor card, transmits the contents of the memory MEM-SEQ along with the corresponding specific codes C_i to the compactor. In effect, the target system re-computes a file of earlier specific coefficients, denoted by F-C-ANT, from the file or table TAB-PRO and the contents of the memory MEM-SEQ, relating to the applications A_1 to A_{k-1}. The file F-C-ANT ensures that each specific code C_i is reciprocally matched with the sequence S_i associated with it for all the applications A_1 to A_{k-1}. Under these conditions in a simplified but not restrictive embodiment, the file F-C-ANT may consist of a file of the same format as said file FEX. In the preferred compacting process illustrated in figure 5, the file F-C-ANT of earlier specific codes is then communicated to the compactor in order to teach the latter.

When compressing a new application with an identification number A_k, the compactor will look for all occurrences of instruction sequences S_i already registered in the file F-C-ANT, i.e. in fact in the table TAB-PRO of the target system for the earlier applications A_1 to A_{k-1}. Whenever an occurrence is found, the compactor system replaces the corresponding sequence of instructions S_i with the specific code C_i for the corresponding specific instruction IS. Once this operation has been completed, the compactor system can then analyse the application bearing identification code A_k and, of course, look for other occurrences with a view to creating additional specific instructions which have not yet been stored. The file F-C-ANT can then be updated. The decompression process described with reference to figure 5 may advantageously be run in particular as a means of compressing either programmes being loaded on the on-board system for the first time or programmes loaded on the on-board system in addition to other existing compacted programmes.

In either of the two cases above, the compacting process proposed by the invention consists in storing the execution table relating to at least one compacted intermediate object code programme, the first of these programmes in the first hypothesis and one or more existing compacted programmes in the second hypothesis, and then reading the stored execution table for every additional intermediate programme and compressing every additional programme, taking account of the specific instructions and codes stored in the execution table as described earlier in the description. Clearly, the compacted intermediate object code programme thus created can not be run except on the target system, which provided the compactor system with the corresponding relevant file F-C-ANT previously.

When running the method of compacting an intermediate object code programme, any on-board system such as a multi-application portable object comprising a microprocessor card, for example, and computing resources such as a microprocessor, a programmable memory, a read only memory and a language interpreter will have, as in figure 2c discussed earlier, at least one set of files stored in the programmable memory 18a for example, in addition to the table TAB-STD of standard codes representing an intermediate object code programme stored at the level of the interpreter.

Accordingly, the corresponding portable object will have at least one compacted intermediate object code programme, i.e. the file FCC illustrated in figure 2c. This file may be made up of an application as described above or may be a function such as a data encryption/decryption function or similar. This compacted intermediate object code file will, of course, consist of a series of specific instruction codes C_1 and standard instruction codes corresponding to the instruction codes of said intermediate object code programme. The specific instruction codes C_i correspond to sequences of successive standard instructions S_i as mentioned earlier in the description.

Furthermore, as illustrated in figure 2c, an execution table allows a reciprocal link to be applied between each specific operating code C_i and the sequence of successive standard instructions S_i associated with this latter. All of these files enable the memory space occupied in the memory to be optimised, in particular the programmable memory 18a of the portable object.

As also illustrated in figure 2c, the execution table has at least one file of successive sequences corresponding to the specific instructions, this file being denoted by the memory MEM-SEQ, and a table, denoted by TAB-PRO, of specific instruction codes and addresses at which these specific instructions are embedded in the file of successive sequences.

The compacted intermediate object code programme is then run as illustrated in figure 4.

A system for compressing an intermediate object code programme enabling the compression method described earlier in the description to be run will now be explained with reference to figures 6a and 6b.

Generally speaking, the compaction method proposed by the invention will be described as a combination of modules, these modules being implemented either by hardware means but preferably by software means, and the data flows between these modules explained.

Figure 6a illustrates the compaction system proposed by the invention, which comprises at least one module A for analysing all the directly executable instructions constituting the intermediate object code programme, shown by COD-OBJ-INT. Generally speaking, the computer file supporting the intermediate object code programme is regarded as a string of octets or character string and the operating mode of the compacting system proposed by this invention will be looked at from the point of view of the corresponding processing applied to the string.

Starting with said string of octets, the analysis module A is able to read the object code programme COD-OBJ-INT, single out and establish a list of all the standard instruction sequences S_i contained in said programme. In figure 6a, the standard instruction sequences S_1, S_{i-1}, S_i, S_{i+1}, ... S_p, are therefore written in symbolic form in a list in accordance with the symbolic notation of the lists. Accordingly, the analysis module A may consist of a sliding window corresponding to a number n_i of octets, this sliding window enabling the analysis of the sequence S_i to be run as mentioned above with reference to table 1 of the description. The sliding window in fact distinguishes between each sequence S_i by scrolling the string of octets relative to said window. Every time the sequence S_i in question occurs, a counting bit BC is issued by the analysis module A.

As also illustrated in figure 6a, the compaction system proposed by the invention also has a module C for counting the number of occurrences in said object code programme of each of the previously mentioned directly executable instruction sequences S_i. The counting module C may be provided in the form of a software module which counts the number of successive bits assigning a value of 1 to said counting bit BC. The counting module C enables the corresponding number of occurrences N_1 ... N_{i-1}, N_i, N_{i+1} ... N_p of each sequence S_1 ...S_{i-1} to S_{i+1} ... S_p and so on to be stored. These can be stored in the form of a list.

Furthermore, as illustrated in figure 6a, a module AL is provided for allocating to at least one sequence of directly executable instructions S_i a specific code C_i associated with this sequence S_i in order to generate a specific instruction, denoted by IS_i in figure 6a, on the basis of a criterion whereby the function of at least the corresponding number of occurrences N_i is higher than a reference value as mentioned earlier in the description.

If the function of at least the number N_i is higher than the function value of the above-mentioned reference value, the module AL will issue a compaction command COM-COMP which may consist of one bit with a corresponding value 1 or 0.

Finally, the compaction system proposed by the invention has a compacting module strictly speaking, COMP, which receives on the one hand the file relating to said intermediate object code programme COD-OBJ-INT and the counting command COM-COMP. The compacting module strictly speaking COMP in effect enables every occurrence of every sequence S_i in said intermediate object code programme, considered as a string of octets, corresponding to a specific instruction IS_i to be replaced by the specific code C_i associated with this sequence of instructions.

As regards the operating mode of the compacting module COMP as such, it should be pointed out that it may have a reading sub-module which uses a sliding window, similar to that of the analysis module, enabling the sequence of standard instructions S_i in said string of octets to be located. In practice, when said sequence of standard instructions S_i is located, as illustrated in figure 6a, the compacting module may have a left-hand partitioning and a right-hand partitioning sub-module for the sequence S_i in question in order to generate a left string, denoted by LS, and a right string, denoted by RS. It may then have a concatenation module which uses the specific code C_i constituting the specific instruction IS_i, so that the corresponding specific code C_i, considered as a string of octets, can be concatenated on the one hand, with the left chain LS for example, after which the unit thus formed is concatenated with the right chain RS, which ensures that the sequence S_i will be replaced by the specific code C_i. The compacting module strictly speaking COMP thus issues a compacted intermediate object code programme denoted by COD-OBJ-INT-COMP in figure 6a. Clearly, the compaction system illustrated in figure 6a enables the compaction process described above to be applied to the set of all the directly executable instruction sequences S_i in question.

In one embodiment which is not restrictive, the allocation module AL may, as illustrated in figure 6b, have a module for computing the number of octets from the length n_i of the instruction sequence S_i, this module being denoted by AL_1 in figure 6b. It may also have a computing module, shown by AL_2, to form the product of this length n_i and the number of occurrences N_i of this sequence S_i of standard instructions. This product, written P_i, is representative of the compression gain for the sequence of directly executable instructions S_i considered.

Furthermore, the allocation module AL may have a module, denoted by AL_3, for comparing this product P_i with a given threshold value, S. The threshold value S may be determined experimentally. It may also be established on the basis of this experimental value in such a way that, for an intermediate object code programme of a given length, it corresponds to a given percentage of this length.

If a negative response comes back from the comparison test run by the module AL_3, the rank i of each directly executable sequence of instructions S_i is incremented by one unit and the new value of i is sent back to the analysis module A on the one hand and the counting module C on the other.

If a positive response comes back from the comparison test run by the module AL_3, a module AL_4 will establish a corresponding specific code C_i and, finally, a module AL_5 will apply a reciprocal link so as to write the specific code C_i and the relevant sequence of directly executable instructions S_i to constitute the specific instruction IS_i.

The module AL_4 may be a compression software module enabling an initial value, for example the value 106 mentioned earlier in the description, to be used as a basis for allocating a corresponding value to the relevant corresponding sequence of instructions S_i. Each specific instruction IS_i may then be written in the form of a corresponding list.

Real-time compression tests on the programmes or applications contained in the microprocessor cards sold by BULL CP8 in France have shown that a compression gain of more than 33% can be obtained, which, when applying the compaction process to three applications on a mobile portable object, essentially represents a gain of one additional application for this type of object.

This compression gain was obtained under substantially normal conditions of usage by the user whilst the slowing-down caused by calling up macro instructions is not substantially more than 10% of the running time if no macro instructions are included, this slowing-down being inherent in successively calling up reading operations at the level of the table TAB-B-PRO and the file MEM-SEQ.

CLAIMS

1.. A method of compacting an intermediate programme consisting of a sequence of standard instructions, used in an on-board system, this on-board system being provided with a memory and a programme language interpreter capable of turning the intermediate programme into instructions of an object code that can be run directly by a microprocessor, said method consisting in:

a) searching through the intermediate programme for identical sequences of successive standard instructions;

b) subjecting the identical sequences of successive instructions to a comparison test to find a function, based on at least the number of occurrences of these sequences in said intermediate programme, that is higher than a reference value and, if the test returns a positive response, for each identical sequence of successive standard instructions which satisfies said test step,

c) a specific instruction is generated by defining a specific operating code and associating this specific operating code with the sequence of successive standard instructions which satisfied said test,

d) replacing each occurrence of each sequence of standard successive instructions in said intermediate programme with said specific operating code associated with it to obtain a compacted intermediate programme, consisting of a series of standard instructions and specific operating codes, and

e) storing in said memory an execution table which enables a reciprocal link to be made between each specific operating code inserted and the sequence of successive standard instructions associated with the latter, which enables the memory space occupied by said compacted intermediate programme to be optimised by storing only one occurrence of said identical sequences of successive standard instructions in said memory.

2. A method as claimed in claim 1, characterised in that said function is also a function of the size of each identical sequence of successive instructions.

3. A method as claimed in claim 1, characterised in that in order to compress a plurality of intermediate programmes, said method also consists in:

- storing the execution table relating to at least one compacted intermediate programme and, for every additional intermediate programme subjected to a compaction process,
- reading said stored execution table and
- running the compaction process for every additional programme, taking account of the specific codes and instructions stored in this execution table.

4. A method of running a compacted intermediate programme obtained by applying the compaction method as claimed in claim 1, and consisting of a succession of standard instructions and specific operating codes stored in the memory of an on-board system, characterised in that it consists in:

- recognising in said memory the existence of a stored execution table containing at least one sequence of successive instructions associated with a specific operating code by means of a reciprocal link;
- calling up a command, via the interpreter, to read the successive standard instructions or specific operating codes of the compacted intermediate programme and, in the presence of a specific operating code:
- retrieving said sequence of successive

instructions associated with said specific operating code from the memory by means of a read instruction and, in the presence of a standard instruction,

- commanding the execution of this instruction by means of a read instruction,

5. A method as claimed in claim 4, characterised in that if a sequence of successive instructions associated with a specific operating code is called up, the current value of a programme counter is incremented in a stack associated with the specific operating codes and a programme pointer points to the first instruction of said sequence of specific instructions, after which, on running an instruction to end the sequence of specific instructions, said programme counter is decremented and the execution process continues starting with the next instruction or specific operating code.

6. A method as claimed in claim 5, characterised in that the stack associated with the specific operating codes and the stack associated with the standard instructions are a single stack.

7. A multi-application on-board system comprising computing resources, a memory and language interpreter capable of turning an intermediate programme into instructions which are directly executable by the computing resources, characterised in that said multi-application on-board system also has at least one table of standard codes constituting said intermediate programme stored at the level of said interpreter;

- at least one compacted intermediate programme constituting an application and consisting of a series of specific instruction codes and standard instruction codes, said specific instruction codes corresponding to sequences of successive standard instructions;

- an execution table enabling a reciprocal link to be made between a specific operating code and the sequence of successive standard instructions associated with the latter, said at least one compacted intermediate programme and said execution table being stored in said memory, enabling the memory space occupied by said compacted intermediate programme to be optimised by storing in said programmable memory only one occurrence of said identical sequences of successive instructions.

8. An on-board system as claimed in claim 7, characterised in that said execution table comprises at least:

- a file of successive sequences corresponding to the specific instructions;

- a table of specific instruction codes and addresses at which these specific instructions are embedded in the table of successive sequences.

9. An on-board system as claimed in claim 8, characterised in that said file of successive sequences corresponding to the specific instructions and said table of specific instruction codes are stored in a programmable memory of said on-board system.

10. A compaction system for an intermediate programme, this intermediate programme consisting of a series of standard instructions which can be executed by a target unit, characterised in that said system comprises at least:

- means for analysing all the standard executable instructions enabling, by means of a reading process, said intermediate programme to distinguish between and establish a list of all the sequences of executable standard instructions contained in this intermediate programme;

- means for counting the number of occurrences in this intermediate programme of each of the sequences of executable standard instructions forming part of this list;

- means for allocating to at least one sequence of executable standard instructions a specific code associated with this sequence of executable standard instructions in order to generate a specific instruction;

- means for replacing, in the programme, each occurrence of this sequence of executable standard instructions with the specific code associated with this sequence of executable standard instructions, representative of said specific instruction, which enables a compacted programme to be generated comprising a succession of executable standard instructions and specific instructions.

11. A system as claimed in claim 10, characterised in that said means for allocating to at least one sequence of executable standard instructions a specific code associated with this sequence of executable standard instructions in order to generate a specific instructions comprises at least:

- means for computing the value of a function based on at least the length of and number of occurrences of this sequence of executable standard instructions, said function being representative of the compression gain for this sequence of executable standard instructions;

- means for comparing the value of this function with a threshold value and, if said comparison returns a positive response,

- means for writing to a file, with a reciprocal link, a specific code and this sequence of executable standard instructions in order to constitute said specific instruction.

09/582939

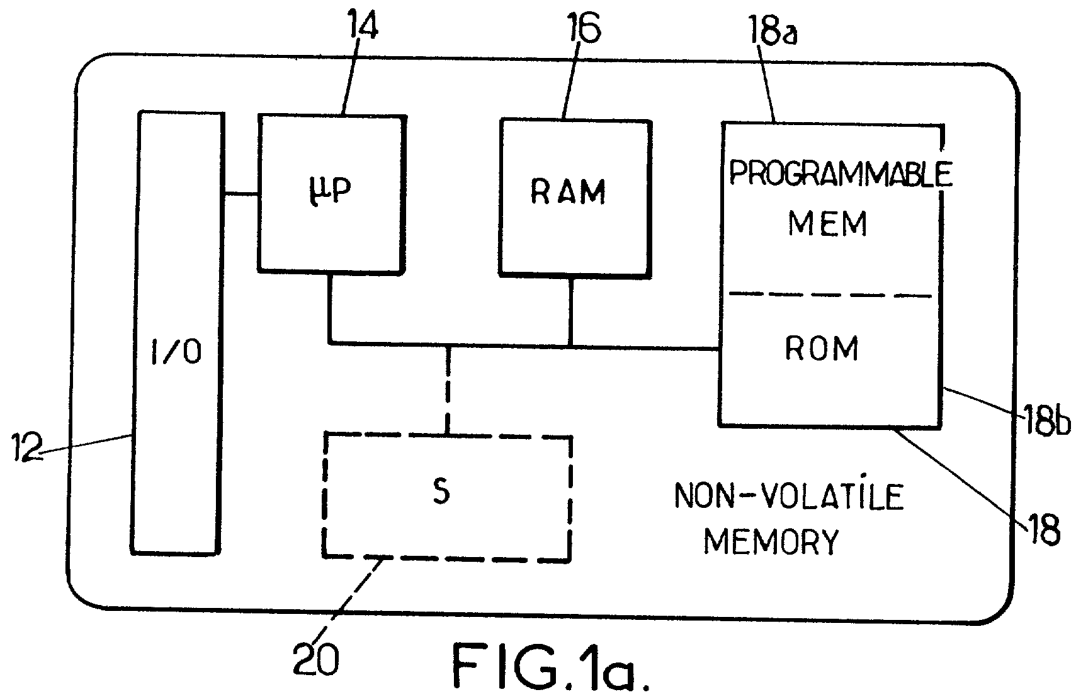


FIG.1a.
(PRIOR ART)

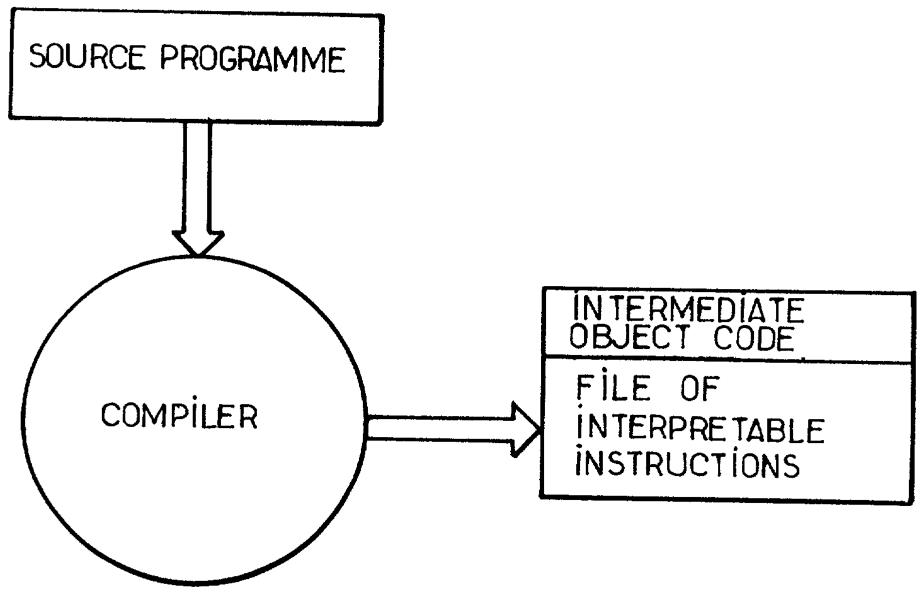
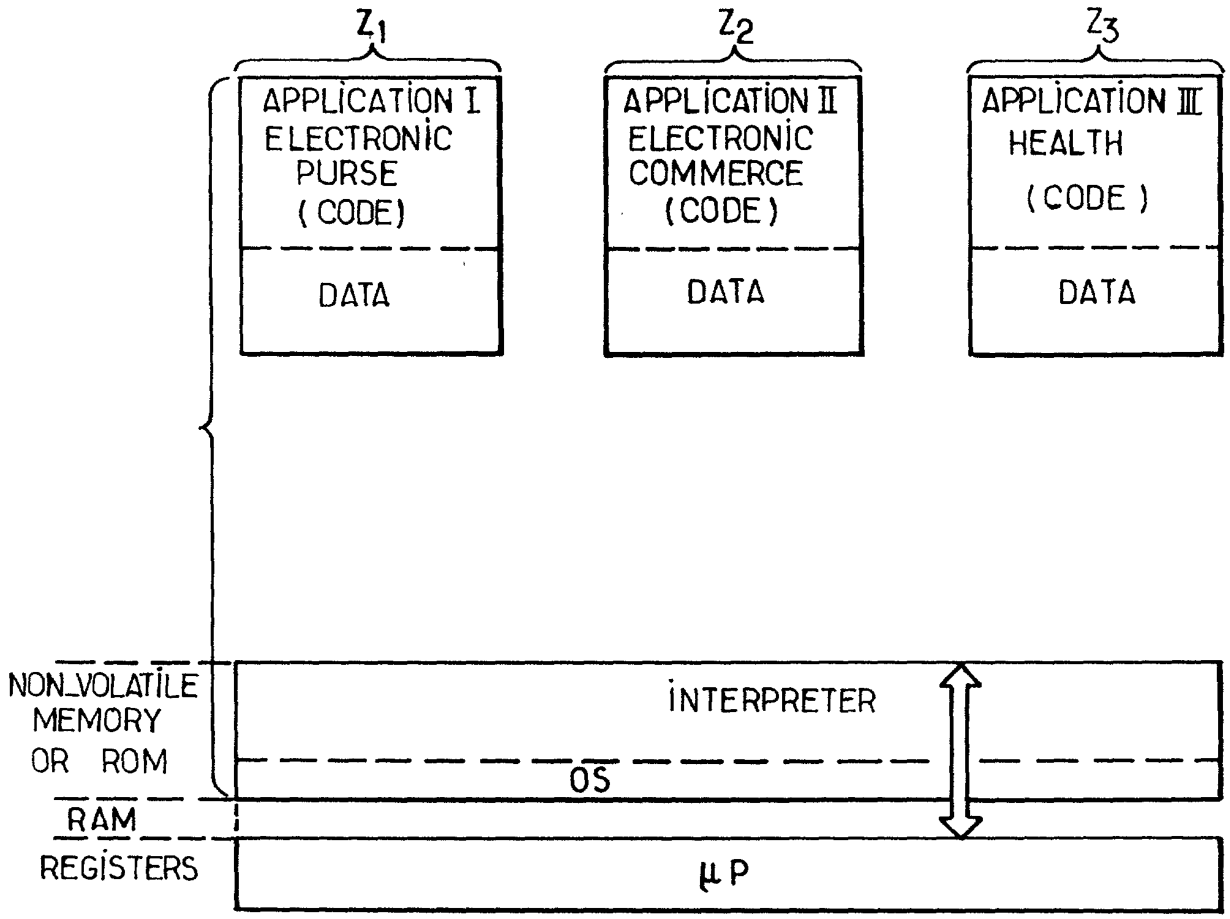


FIG.1b.
(PRIOR ART)

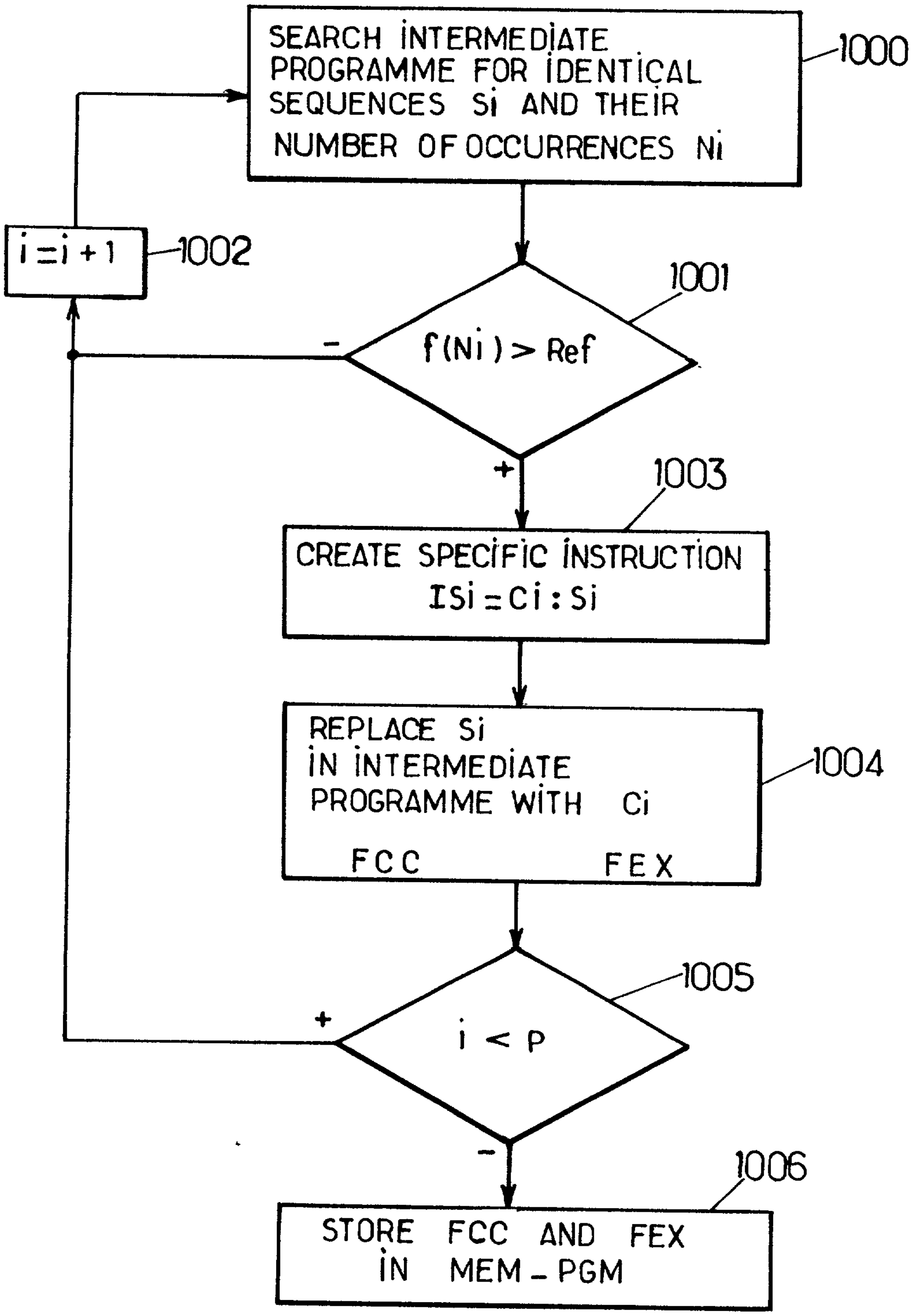
09/582939



Z1
Z2
Z3
APPLICATION I ELECTRONIC PURSE (CODE)
DATA
APPLICATION II ELECTRONIC COMMERCE (CODE)
DATA
APPLICATION III HEALTH (CODE)
DATA
NON_VOLATILE MEMORY OR ROM
INTERPRETER
OS
RAM
REGISTERS
µP

FIG.1c.
(PRIOR ART)

09/582939



SEARCH INTERMEDIATE PROGRAMME FOR IDENTICAL SEQUENCES Si AND THEIR NUMBER OF OCCURRENCES Ni
1000
i := i + 1
1002
f(Ni) > Ref
1001
-
+
1003
CREATE SPECIFIC INSTRUCTION ISi = Ci : Si
REPLACE Si IN INTERMEDIATE PROGRAMME WITH Ci
FCC
FEX
1004
i < P
1005
+
-
1006
STORE FCC AND FEX IN MEM-PGM

FIG.2a.

FIG.2b.

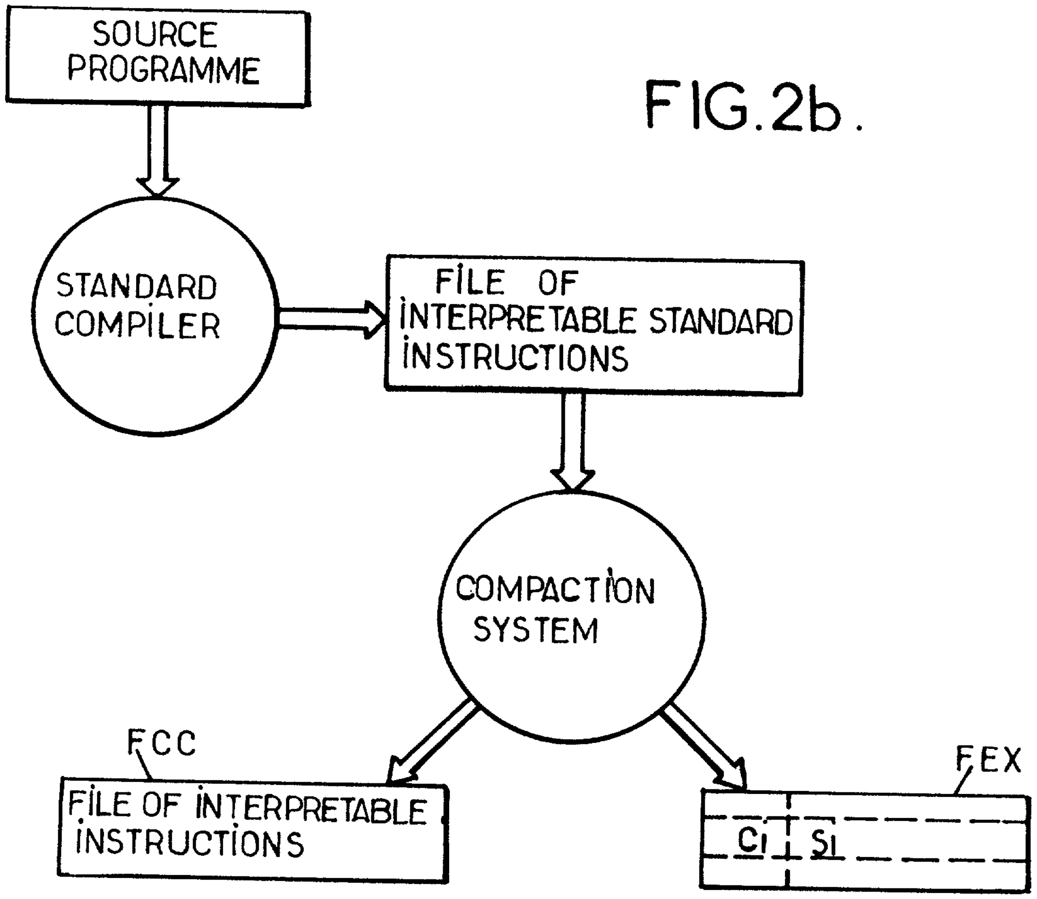
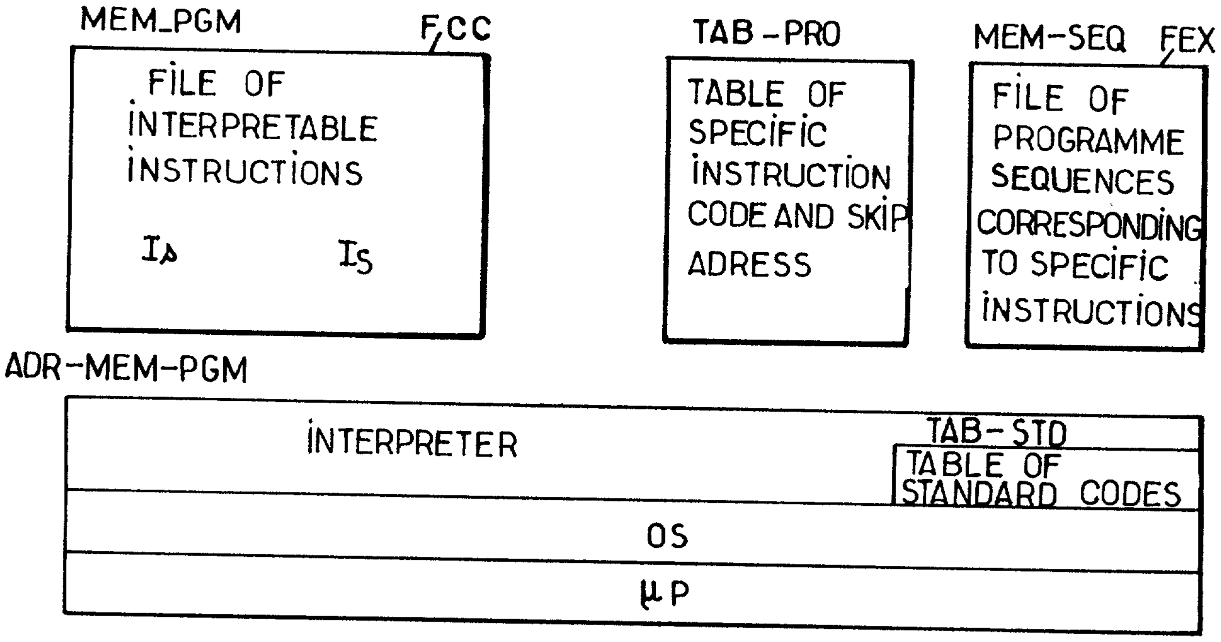
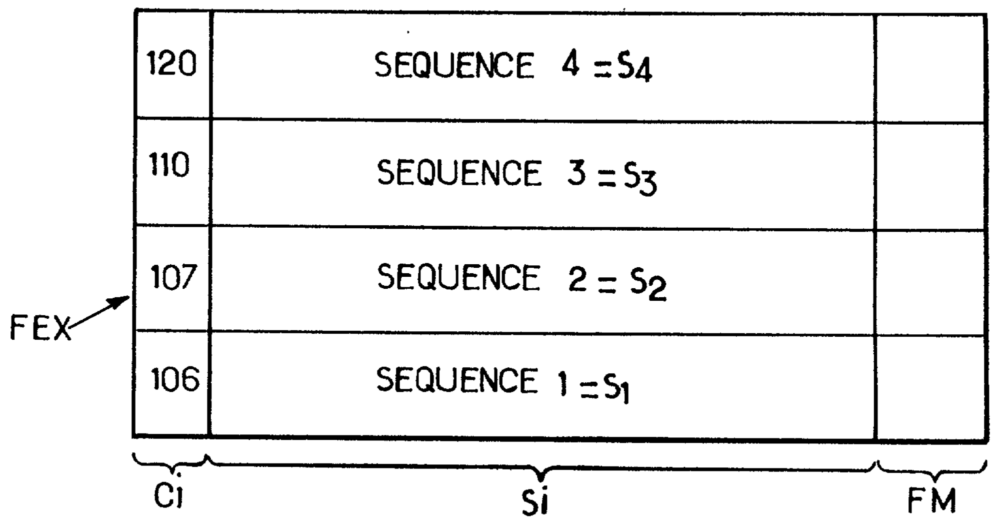

SOURCE PROGRAMME
STANDARD COMPILER
FILE OF INTERPRETABLE STANDARD INSTRUCTIONS
COMPACTION SYSTEM
FCC
FILE OF INTERPRETABLE INSTRUCTIONS
FEX
Ci
Si

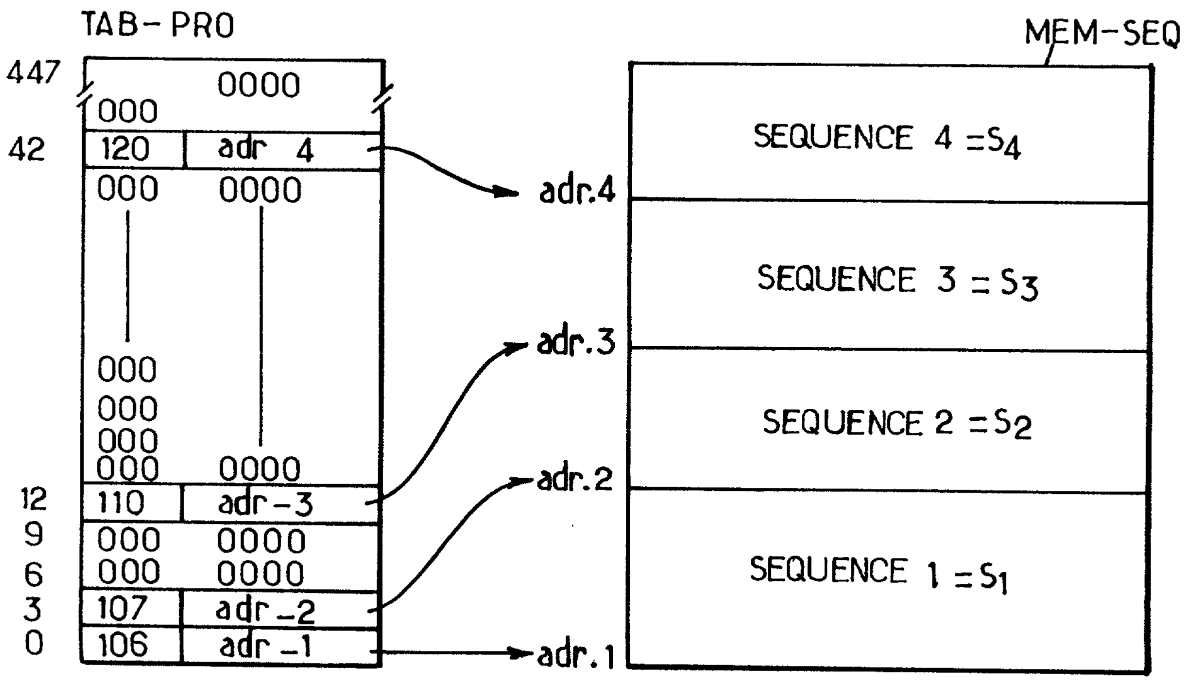
FIG.2c.


MEM_PGM
FCC
FILE OF INTERPRETABLE INSTRUCTIONS
IA
IS
TAB-PRO
TABLE OF SPECIFIC INSTRUCTION CODE AND SKIP ADRESS
MEM-SEQ
FEX
FILE OF PROGRAMME SEQUENCES CORRESPONDING TO SPECIFIC INSTRUCTIONS
ADR-MEM-PGM
INTERPRETER
TAB-STD
TABLE OF STANDARD CODES
OS
μP



120
SEQUENCE 4 = S4
110
SEQUENCE 3 = S3
107
SEQUENCE 2 = S2
106
SEQUENCE 1 = S1
FEX
Ci
Si
FM

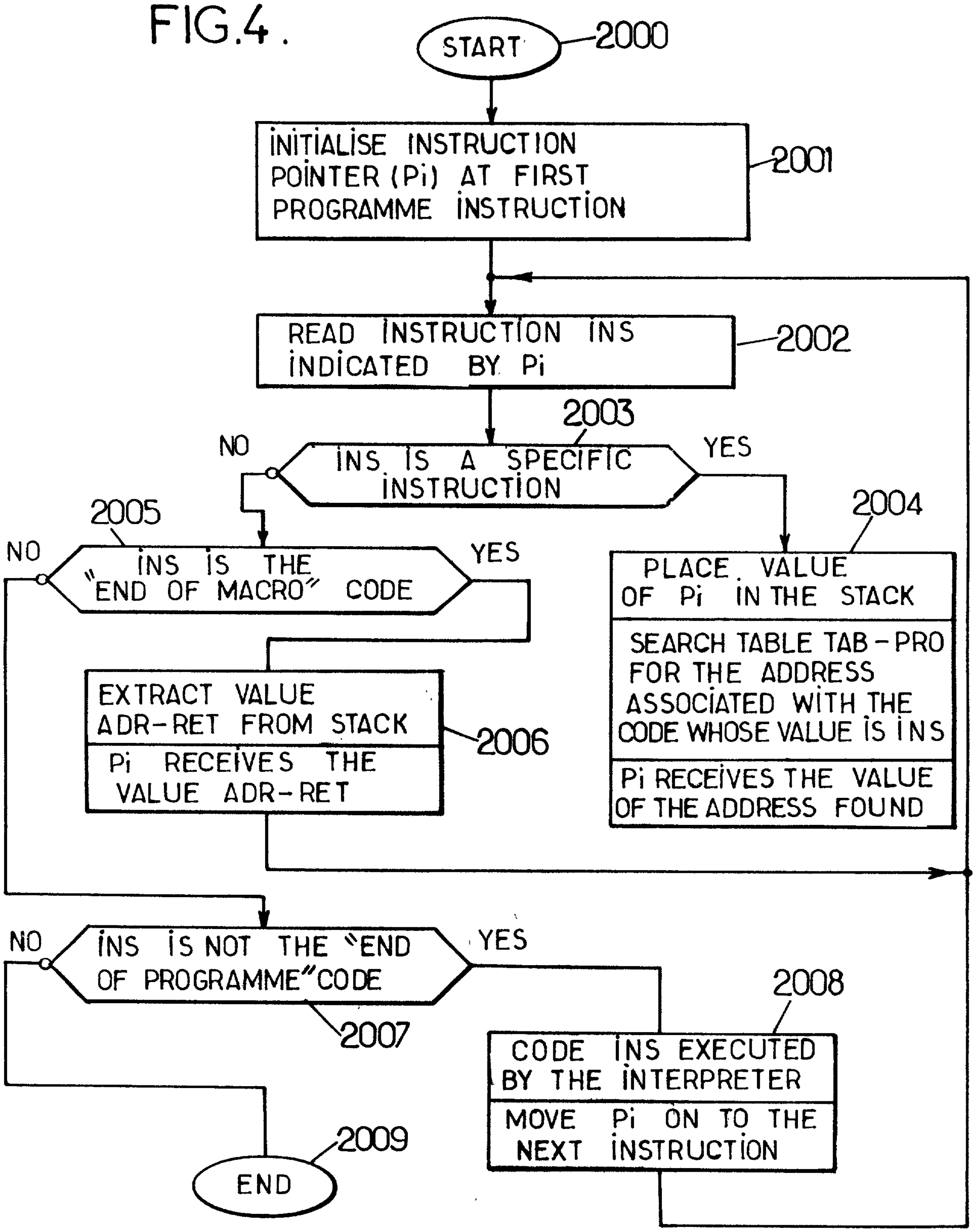
FIG.3a.



TAB-PRO
447
0000
000
42
120
adr 4
000
0000
000
000
000
000
0000
12
110
adr-3
9
000
0000
6
000
0000
3
107
adr-2
0
106
adr-1
adr.4
adr.3
adr.2
adr.1
MEM-SEQ
SEQUENCE 4 = S4
SEQUENCE 3 = S3
SEQUENCE 2 = S2
SEQUENCE 1 = S1

FIG.3b.

FIG.4.


START
2000
INITIALISE INSTRUCTION POINTER (Pi) AT FIRST PROGRAMME INSTRUCTION
2001
READ INSTRUCTION INS INDICATED BY Pi
2002
2003
NO
INS IS A SPECIFIC INSTRUCTION
YES
2004
PLACE VALUE OF Pi IN THE STACK
SEARCH TABLE TAB-PRO FOR THE ADDRESS ASSOCIATED WITH THE CODE WHOSE VALUE IS INS
Pi RECEIVES THE VALUE OF THE ADDRESS FOUND
2005
NO
INS IS THE "END OF MACRO" CODE
YES
EXTRACT VALUE ADR-RET FROM STACK
Pi RECEIVES THE VALUE ADR-RET
2006
NO
INS IS NOT THE "END OF PROGRAMME" CODE
YES
2007
2008
CODE INS EXECUTED BY THE INTERPRETER
MOVE Pi ON TO THE NEXT INSTRUCTION
2009
END

09/582939

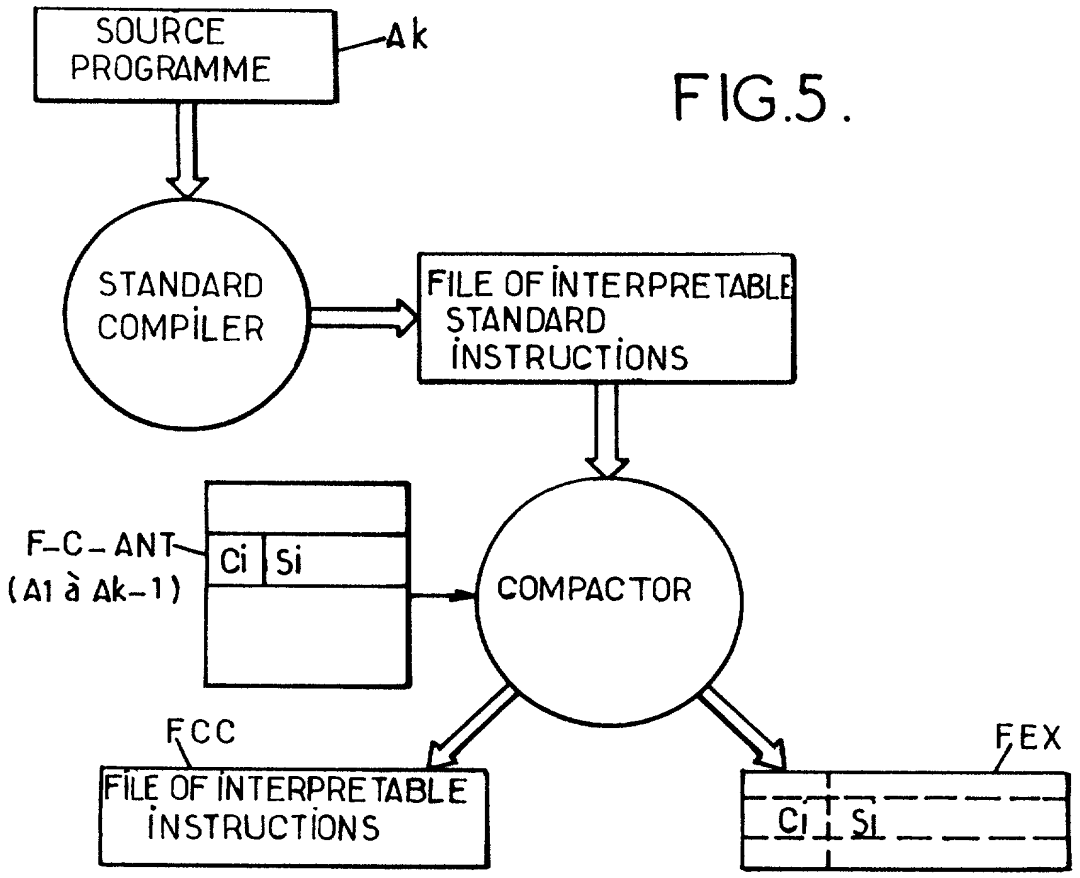

SOURCE PROGRAMME
Ak
STANDARD COMPILER
FILE OF INTERPRETABLE STANDARD INSTRUCTIONS
F-C-ANT
(A1 à Ak-1)
Ci
Si
COMPACTOR
FCC
FILE OF INTERPRETABLE INSTRUCTIONS
FEX
Ci
Si

FIG.5.

09/582939

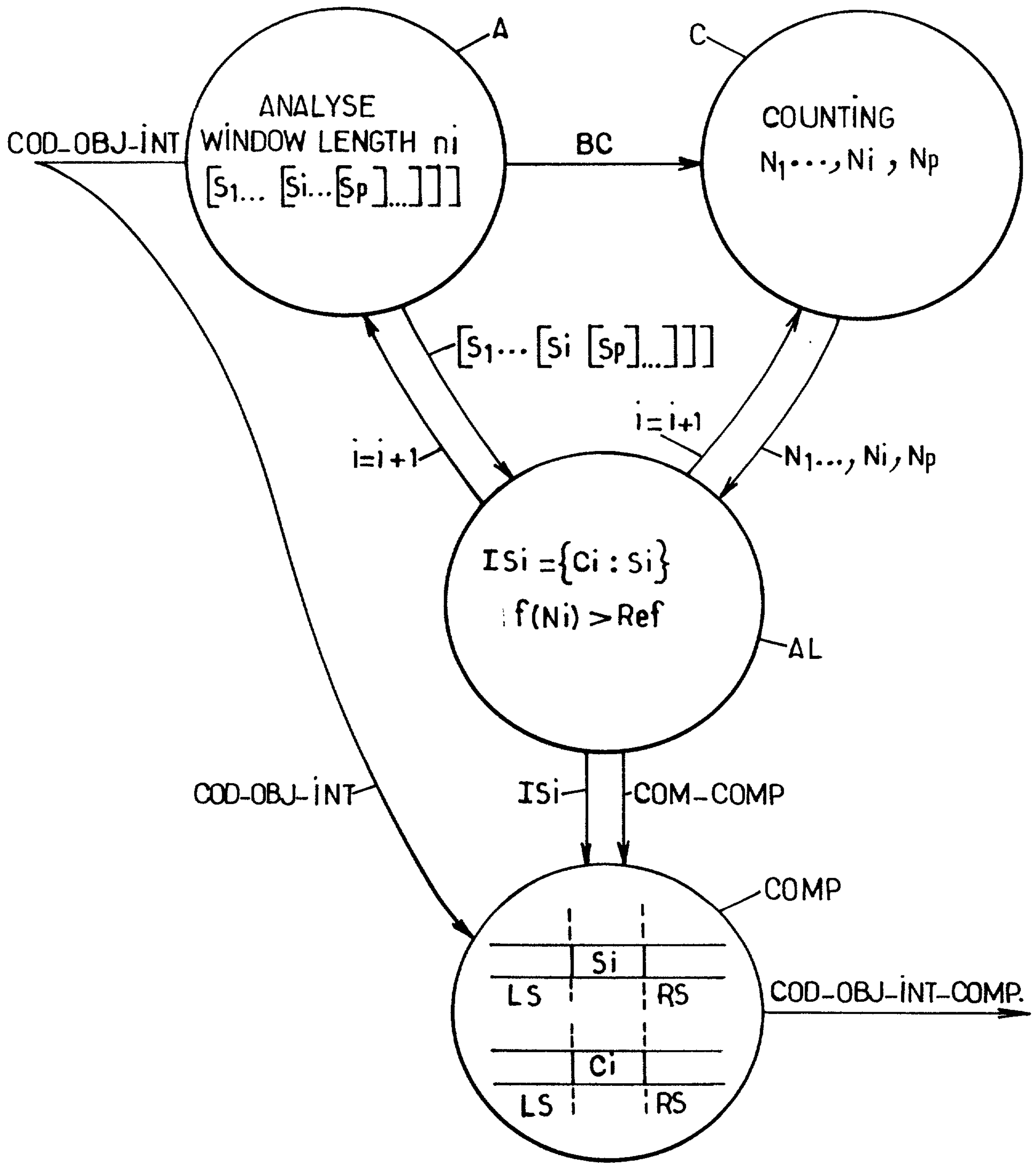
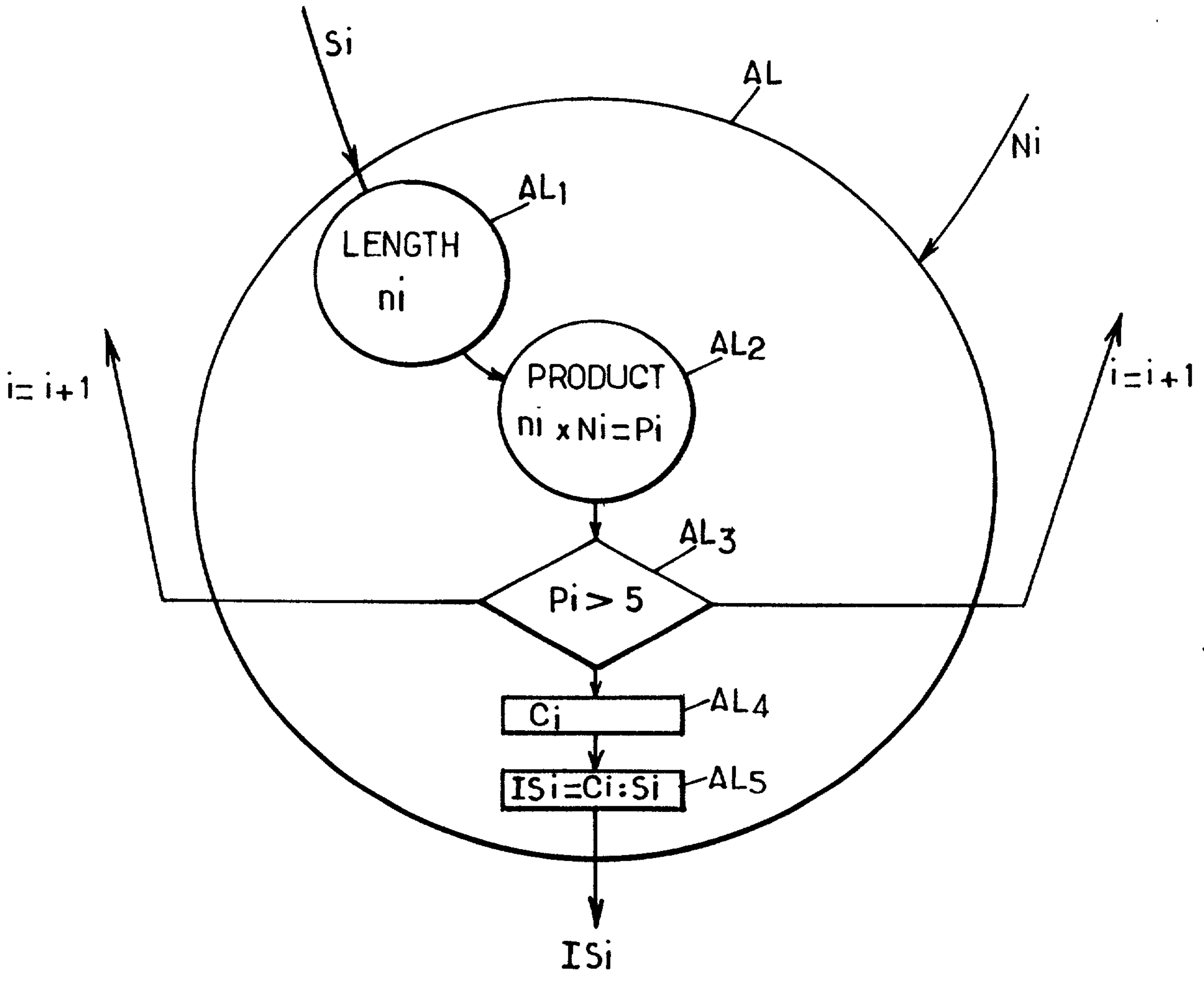


FIG. 6a.

FIG.6b.


Si
AL
Ni
AL1
LENGTH ni
AL2
PRODUCT ni x Ni = Pi
i:= i+1
i:=i+1
AL3
Pi > 5
AL4
Ci
AL5
ISi=Ci:Si
ISi

Express Mail Label No.

Docket No.

Declaration and Power of Attorney For Patent Application

English Language Declaration

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled A DATA COMPACTION METHOD FOR AN INTERMEDIATE OBJECT CODE PROGRAMME EXECUTABLE IN AN ON-BOARD SYSTEM PROVIDED WITH DATA PROCESSING RESOURCES AND CORRESPONDING ONBOARD SYSTEM WITH MULTIPLE APPLICATIONS.

the specification of which

(check one)

☐ is attached hereto.

☒ was filed on 6th July 2000 as United States Application No. or PCT International Application Number 09/582 939

and was amended on __________ (if applicable)

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119(a)-(d) or Section 365(b) of any foreign application(s) for patent or inventor's certificate, or Section 365(a) of any PCT International application which designated at least one country other than the United States, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate or PCT International application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application(s)

			Priority Not Claimed
98 14012	FRANCE	6th November 1998	☐
(Number)	(Country)	(Day/Month/Year Filed)	
			☐
(Number)	(Country)	(Day/Month/Year Filed)	
			☐
(Number)	(Country)	(Day/Month/Year Filed)	

I hereby claim the benefit under 35 U.S.C. Section 119(e) of any United States provisional application(s) listed below:

(Application Serial No.)	(Filing Date)
(Application Serial No.)	(Filing Date)
(Application Serial No.)	(Filing Date)

I hereby claim the benefit under 35 U. S. C. Section 120 of any United States application(s), or Section 365(c) of any PCT International application designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. Section 112. I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, C. F. R., Section 1.56 which became available between the filing date of the prior application and the national or PCT International filing date of this application:

FR99/02696	4th November 1999	
(Application Serial No.)	(Filing Date)	(Status) ~~(patented~~, pending, ~~abandoned)~~
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)

09582369 . 102700

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. *(list name and registration number)*

9

Michael L. Kenaga	34,639
William T. Rifkin	26,501
Mark I. Feldman	26,880
James P. Ryther	20,424
Mary Spalding Burns	32,116
R. Blake Johnston	41,097
Thomas W. Ryan	43,072
Tracey R. Thomas	38,633
David J. Richter	26,221

Send Correspondence to: Michael L. Kenaga
RUDNICK & WOLFE
P.O. Box 64807
Chicago, Illinois 60664-0807

Direct Telephone Calls to: *(name and telephone number)*
Michael L. Kenaga, (312) 368-8937

1-00

Full name of sole or first inventor Ulrik Pagh SCHULTZ	
Sole or first inventor's signature SCHULTZ, Ulrik [illegible]	Date 9th october 2000
Residence 35000 RENNES (FRANCE) FRX	
Citizenship FRENCH	
Post Office Address 37 rue Saint Georges - 35000 RENNES (FRANCE)	

2-00

Full name of second inventor, if any Gilles MULLER	
Second inventor's signature Gilles Muller [illegible]	Date 9th october 2000
Residence 35690 ACIGNE (FRANCE) FRX	
Citizenship FRENCH	
Post Office Address 12 rue de la Pommeraie - 35690 ACIGNE (FRANCE)	

3-00 Full name of third inventor, if any Charles CONSEL	
Third inventor's signature Charles CONSEL [illegible]	Date 9th october 2000
Residence 35240 RETIERS (FRANCE) FRX	
Citizenship FRENCH	
Post Office Address 10 boulevard Alphonse Richard - 35240 RETIERS (FRANCE)	

4-00 Full name of fourth inventor, if any Lars CLAUSEN	
Fourth inventor's signature Lars Clausen [illegible]	Date 9th october 2000
Residence URBANA, IL 61801 (USA) IL	
Citizenship DANISH	
Post Office Address 1010 W. Green Street, Apt 67 - URBANA, IL 61801 (USA)	

5-00 Full name of fifth inventor, if any Christian GOIRE	
Fifth inventor's signature [illegible]	Date 9th october 2000
Residence 78340 LES CLAYES SOUS BOIS (FRANCE) FRX	
Citizenship FRENCH	
Post Office Address 8 allée du Mail - 78340 LES CLAYES SOUS BOIS (FRANCE)	

Full name of sixth inventor, if any	
Sixth inventor's signature	Date
Residence	
Citizenship	
Post Office Address	